Assignment 4

Bhanu Verma (903151012)

**Part 1: Branch and Bound**

**Answer 1**: To start with let’s define the given problem, MINIMUM SET COVER, first. Given a set of elements (number of elements, ), called the universe , and total number of sets, , whose union equals to universe, we need to find the smallest sub-collection of sets, , whose union is equal to universe .

We need to devise a branch and bound algorithm for Minimum Set Cover. To design our branch and bound, we will use an array of size (number of sets), 123m, to store our partial solutions. For each subset till , we evaluate if i is part of solution (i becomes 1) or not part of solution (i becomes 0). If we haven’t looked at a subset yet, then i will contain -1.

**a).** What is a sub-problem for branch and bound algorithm?

**Sol a).** If we have a partial solution 123m then as per our above definition, let be the subsets for the given partial solution i.e. sets that are included in the set cover. can be formally defined as:

For the partial solution, we have as the set of unique elements covered by subsets, in the partial solution. Now, our sub-problem can be defined as smaller set cover problem with a reduced universe i.e. universe obtained by removing the elements that have been covered at-least once by the subsets in the partial solution. Sets for our sub-problem will be the sets, for which no decision has been made yet i.e. all the sets for which i = -1. We have defined out sub-problem in terms of and .

**b).** How do you choose a sub-problem to expand?

**Sol b).** Intuitively if think about this, to minimize our number of sets in the solution, at each step we would need to cover the most number of elements that we can. Another way of doing this is picking a sub-problem which leaves us with minimum number of uncovered elements,

**c).** How do you expand a sub-problem?

**Sol c).** Expanding a sub-problem given our solution is pretty straight-forward. We select all elements for which i = -1 and branch out two sub-problems for each element, one each for i = 1 and i = 0.

**d).** What is an appropriate lower-bound?

**Sol d).** An appropriate lower-bound should be the number of subsets used so far in the partial solution.

The choices that we have made above should work well on typical instances of the problem because we have built a generic solution approach for the given problem and have not based our solution on any particular instance.

**Part 2: Outline a simple greedy heuristic for the SET COVER problem, and explain why it finds a valid solution and its running time.**

**Answer 2:**

A simple greedy heuristic can be to add that subset to our SET COVER solution that covers the maximum number of remaining uncovered elements from the universe.

Here is the pseudo code for the algorithm:

123n

123mi

Please note that it is safe to assume that there is some subset of subsets , union for which covers the universe

This algorithm will always find a valid solution (if one exists) because it does not stop iterating till all the elements of the given universe have been visited.

**Time complexity:**

First line inside the while loop computes intersection which can take as we can have at-most elements and sets. Next two lines in the while loop requires linear operations in terms of and , so out time complexity is still Final step is to compute the number of iterations of the while loop. We see that while loop runs till becomes an empty set. If our subsets are such that one elements get removed from one at a time, then while loop runs for number of iterations, otherwise it runs for number of iterations. So, our complexity becomes .

**Part 3: Local Search**

**a).** What could be a possible scoring function for such candidate solutions?

**Sol a).** Our solution tries to maximize the number of elements with as few subsets as possible. We will need to capture this criterion in our scoring function. One possible way of doing this can be building a scoring function by **calculating the number of elements captured by the subsets of a candidate solution and then dividing it by the number of sub-sets it contains**. Such a scoring function will favor those solutions which accumulate the most of elements from the universe with minimal of subsets.

**b).** What would be a Neighborhood (or Moves) you would consider using for your local search to move from one candidate solution to other 'nearby' solutions? How many potential neighbors can a candidate solution have under your Neighborhood (using Big-Oh)?

**Sol b).** A very good criterion for selecting a neighborhood can be to iteratively remove or add subsets, one at a time. In this way, our candidate solutions will differ by one subset. Addition will be for cases when we have to add a new subset as that subset contains elements which are not covered by other subsets in the partial solution so far. Removal will be for cases when we see a subset that covers all the elements covered by a particular subset in the partial solution and covers extra elements on top of that.

As for the terminology that we have used above, we can have at-most number of subsets, so a candidate solution can have potential neighbors.

**c).** Why would you consider adding Tabu Memory and what would be remembered in your Tabu Memory?

**Sol c).** Tabu memory is used for facilitating Tabu search. Tabu search is used for enhancing the performance of local search by relaxing the basic rule of local search i.e. accepting worsening moves or neighbors if no better neighbors are available and additionally we use prohibitions (done by using Tabu memory) to stop the search from coming back to already visited solutions. This is usually done **to prevent your search from getting stuck at local minimum**.

A very simple idea can be to keep a certain number of visited (added or removed) subsets, say , in the memory for last solutions and this memory can then be used to choose candidate solutions which contains subsets that have not already been visited.